Swiss Army Cryptographic Toolset for Beginners

Bhavik Naik | November 12, 2019

Product Overview

The Swiss Army cryptographic toolset for beginners is a great way to be introduced into the world of cryptography! It features an interactive, command line interface that lets the user interact with the program. At the same time, the user learns more about the cryptography tools that we use in our modern world.

About

This toolset is created in Python 3.7 and it includes 3 different cryptographic tasks. It incorporates The Advanced Encryption Standard (AES), the Secure hashing algorithm (SHA256), and elliptic curve cryptography (ECC).

Installation

To install this program and make it run on your computer, you would require these steps:

1. Install Python using: https://www.python.org/downloads/
   - 1.1. Make sure that you check the checkbox to add python to your environment variables so that you can run python from your command prompt.
   - 1.2. Also make sure that 'pip' is installed as well and added to your environment variables, so that you can proceed after the next step.
2. Install the required cryptography.io libraries that are needed to run the 3 different types of cryptographic tasks. Without this step, the program will not run. Install using: `pip install cryptography`
3. After downloading the file that includes the python code:
   - 3.1. Open your command prompt

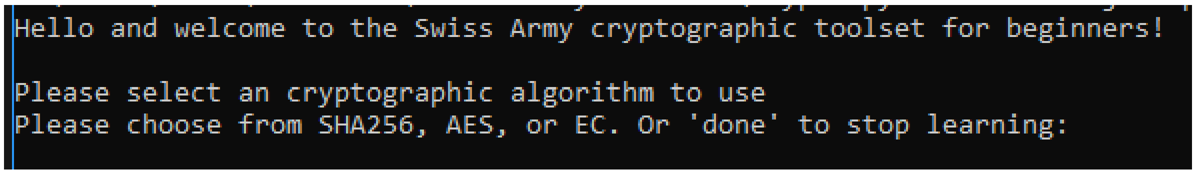
3.2. Navigate to your folder where you saved the file, using the `cd` command

3.3. Type `python [filename].py`

And that's it! If you followed the steps, you should be able to run the program and learn about cryptography tasks.

Operation of Program

If you followed the installation steps, and you got the program to run on your computer, you will see the following text:



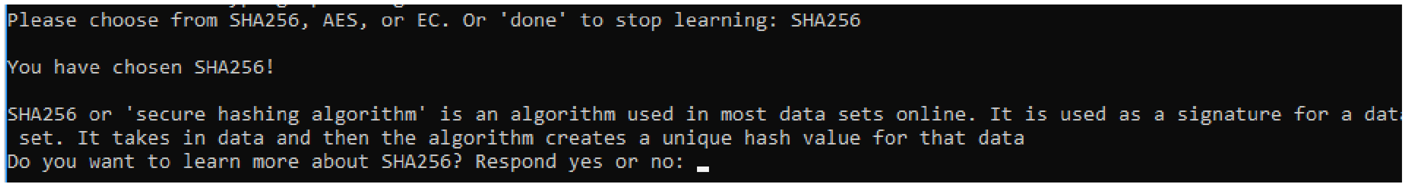
```
Hello and welcome to the Swiss Army cryptographic toolset for beginners!

Please select an cryptographic algorithm to use
Please choose from SHA256, AES, or EC. Or 'done' to stop learning:
```

The program will ask for input. You could either type SHA256, AES, EC, or done. All of these are case sensitive. Selecting the first three will take you to the respective area which teaches you more about the algorithms and goes through a demo with you for each one.

Typing done ends the program. If you input any other text, you will get to try again.

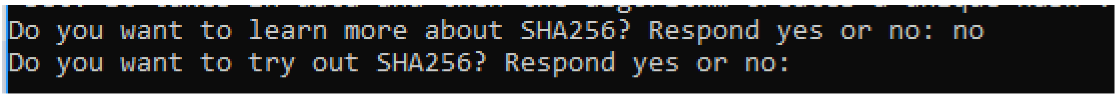
Operations for each algorithm

Typing SHA256 into the command line above yields the following message:



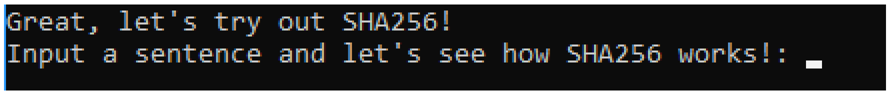
```
Please choose from SHA256, AES, or EC. Or 'done' to stop learning: SHA256

You have chosen SHA256!

SHA256 or 'secure hashing algorithm' is an algorithm used in most data sets online. It is used as a signature for a dat
 set. It takes in data and then the algorithm creates a unique hash value for that data
Do you want to learn more about SHA256? Respond yes or no:
```

It shows a brief overview on what SHA256 is and it asks you if you want to learn more about the algorithm. Typing 'yes' outputs more text, outlining how the algorithm works. Typing 'no' yields this message. This stays the same no matter which algorithm that you chose.



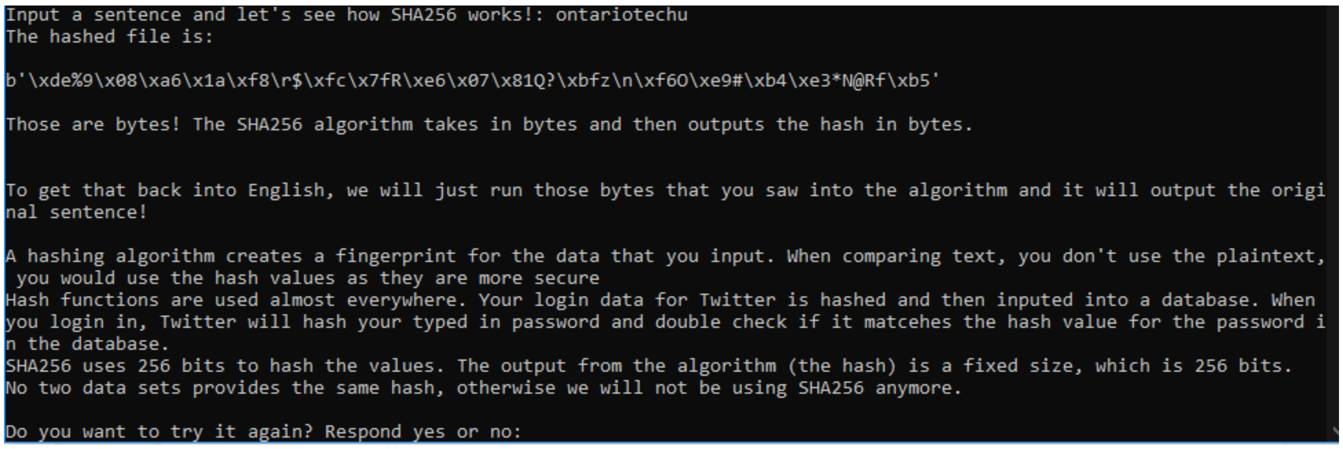
```
Do you want to learn more about SHA256? Respond yes or no: no
Do you want to try out SHA256? Respond yes or no:
```

Typing ‘no’ goes back to the main menu, allowing you to switch algorithms or exit the program using the ‘done’ keyword. Typing ‘yes’ advances to the demo area. This is what you will see when you reach the demo area for each algorithm (algorithm name will vary).



```
Great, let's try out SHA256!
Input a sentence and let's see how SHA256 works!: _
```

These algorithms will make you input a sentence, to showcase the output and end result for each cryptography task. Typing in a sentence:



```
Input a sentence and let's see how SHA256 works!: ontariotechu
The hashed file is:

b'\xde%9\x08\xa6\x1a\xf8\r$\xfc\x7fR\xe6\x07\x81Q?\xbfz\n\xf6O\xe9#\xb4\xe3*N@Rf\xb5'

Those are bytes! The SHA256 algorithm takes in bytes and then outputs the hash in bytes.


To get that back into English, we will just run those bytes that you saw into the algorithm and it will output the origi
nal sentence!

A hashing algorithm creates a fingerprint for the data that you input. When comparing text, you don't use the plaintext,
 you would use the hash values as they are more secure
Hash functions are used almost everywhere. Your login data for Twitter is hashed and then inputed into a database. When
you login in, Twitter will hash your typed in password and double check if it matcehes the hash value for the password i
n the database.
SHA256 uses 256 bits to hash the values. The output from the algorithm (the hash) is a fixed size, which is 256 bits.
No two data sets provides the same hash, otherwise we will not be using SHA256 anymore.

Do you want to try it again? Respond yes or no:
```

This output will vary depending on the algorithm but all of them will outline what is happening and what the output was based on the message that you put. You can see the bytes output above and then an explanation on what happened.

Finally at the end of the text, there will be another input field asking you if you would want to redo the demo with another example text to see the output change, depending on the algorithm. Typing ‘yes’ will make you input text again for that algorithm. Typing ‘no’ will return you back to the main page where you can choose another algorithm to learn about or quit the program using the ‘done’ keyword.

I hope you enjoy learning about SHA256, AES, and the elliptic curve algorithms! The source code for the program is located below in the appendix.

Appendix

```
# Bhavik Naik
# Final Project
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes
import os
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.fernet import Fernet
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import ec

def SHA256():
    sentence = input("Input a sentence and let's see how SHA256 works!: ")
    digest = hashes.Hash(hashes.SHA256(), backend=default_backend())
    digest.update(bytes(sentence, encoding='utf8'))
    finalize = digest.finalize()
    print("The hashed file is:\n")
    print(bytes(finalize))
    print("\nThose are bytes! The SHA256 algorithm takes in bytes and then outputs the hash in
bytes.\n")
    print(" ")
    print("To get that back into English, we will just run those bytes that you saw into the
algorithm and it will output the original sentence!\n")
    print("A hashing algorithm creates a fingerprint for the data that you input. When
comparing text, you don't use the plaintext, you would use the hash values as they are more
secure")
    print("Hash functions are used almost everywhere. Your login data for Twitter is hashed
and then inputted into a database. When you login in, Twitter will hash your typed in password
and double check if it matches the hash value for the password in the database.")
    print("SHA256 uses 256 bits to hash the values. The output from the algorithm (the hash)
is a fixed size, which is 256 bits.")
    print("No two data sets provides the same hash, otherwise we will not be using SHA256
anymore.\n")
    response = input("Do you want to try it again? Respond yes or no: ")
    if response[0] == "y" or response[0] == "Y":
        return 1;
    return 0

def AES():
    sentence = input("Input a sentence and let's see how AES works!: ")
    backend = default_backend()
    key = Fernet.generate_key()
    keys = os.urandom(32)
    iv = os.urandom(16)
    cipher = Cipher(algorithms.AES(keys), modes.CBC(iv), backend=backend)
    encryptor = cipher.encryptor()
    f = Fernet(key)
    ct = encryptor.update(b"a secret message") + encryptor.finalize()
    token = f.encrypt(bytes(sentence, encoding='utf8'))
    decryptor = cipher.decryptor()
    decryptor.update(ct) + decryptor.finalize()
    print(" ")
    print("CIPHERTEXT: " + str(token))
    print(" ")
    print("DECRYPTED: " + str(f.decrypt(token)))
    print(" ")
     print("Again, those are bytes that we are getting back. But, unlike those that are found
on the SHA256 output, we are changing it back into readable text.")
      print("The ciphertext is the text that is being calculated using the AES algorithm.
Basically it is making the text harder to read by applying a filter on it.")
    print("Notice the decrypted text, everything that you inputted is after the 'b'' letter.")
      print("We generated a random number and used that as a key. The key is what the AES
algorithm uses to figure out how much it is needed to change the text")
     print("For example, if we take the text 'ABC' and give a key of 1, the output would be
CAB, moving everything by 1 over. But if we give it a key of 2, it would be BCA. Notice the
change, it is the same thing, only in this case, its random!\n")
     print("AES is a symmetric algorithm which means that it uses the same key to encrypt and
decrypt the text.")
     print("AES creates blocks of data from the user inputted text. It then performs numerous
transformations to this blocks such as flipping them and shifting the rules. This is based
from the keys. The algorithm then will note the transformations that it did it the blocks and
then it will create the ciphertext.\n")

    response = input("Do you want to try it again? Respond yes or no: ")
    if response[0] == "y" or response[0] == "Y":
        return 1;
    return 0

def EC():
     print("Elliptic curve cryptography basically takes the message and signs it so that it is
secure.")
    message = input("Write a message to sign: ")
    message = bytes(message, encoding='utf8')
    private_key = ec.generate_private_key(
        ec.SECP384R1(), default_backend()
        )
    signature = private_key.sign(
        message,
        ec.ECDSA(hashes.SHA256())
        )
    print("We successfully signed the message using the Elliptic Curve algorithm!")
     print("To verify that the message is still signed with the private key, we will run a command that will make sure that it is secure.")
    public_key = private_key.public_key()
    public_key.verify(signature, message, ec.ECDSA(hashes.SHA256()))
    print("Okay! We have double checked that the message that you entered is in fact safe!")
     print("Elliptic curve cryptography is slowly becoming the next big thing in cryptography. It is the next wave of asymmetric algorithms that sign messages. But unlike other popular algorithms, Elliptic curve cryptography works with less bits, meaning that it is more efficient")
     print("It uses the math behind Elliptic curves to generate a public and private key. Each intersection of 3 points in an elliptic curve signifies a number that could be used as a key for either the public or private key.")
     print("A great analogy that the Universal Security study suggests is the competition to Elliptic curve cryptography (RSA), to break a 228 bit key would take less energy to boil a teaspoon of water. Elliptic curve cryptography, to break a 228 bit key, would take more energy to boil all the water on Earth.")
    print("This shows how powerful Elliptic curve cryptography is.\n")
    response = input("Do you want to try it again? Respond yes or no: ")
    if response[0] == "y" or response[0] == "Y":
        return 1;
    return 0

print("Hello and welcome to the Swiss Army cryptographic toolset for beginners!\n")
while True:
    print("Please select an cryptographic algorithm to use");
    inputAlgorithm = input("Please choose from SHA256, AES, or EC. Or 'done' to stop learning: ");
    if (inputAlgorithm == "done"):
        print("Thank you for learning about the cryptographic systems! See you next time")
        break
    elif (inputAlgorithm == "SHA256"):
        print("\nYou have chosen SHA256!\n");
          print("SHA256 or 'secure hashing algorithm' is an algorithm used in most data sets online. It is used as a signature for a data set. It takes in data and then the algorithm creates a unique hash value for that data")
        response = input("Do you want to learn more about SHA256? Respond yes or no: ")
        if response[0] == "y" or response[0] == "Y":
            print("A hashing algorithm creates a fingerprint for the data that you input. When comparing text, you don't use the plaintext, you would use the hash values as they are more secure")
             print("Hash functions are used almost everywhere. Your login data for Twitter is hashed and then inputted into a database. When you login in, Twitter will hash your typed in password and double check if it matches the hash value for the password in the database.")
```

```
                print("SHA256 uses 256 bits to hash the values. The output from the algorithm
(the hash) is a fixed size, which is 256 bits.")
                print("No two data sets provides the same hash, otherwise we will not be using
SHA256 anymore.")
            response3 = input("Do you want to try out SHA256? Respond yes or no: ")
            if response3[0] == "y" or response[0] == "Y":
                print("Great, let's try out SHA256!")
                returnS = SHA256()
                while returnS == 1:
                    returnS = SHA256()
            else:
                continue
        else:
            response2 = input("Do you want to try out SHA256? Respond yes or no: ")
            if response2[0] == "y" or response[0] == "Y":
                print("Great, let's try out SHA256!")
                returnS = SHA256()
                while returnS == 1:
                    returnS = SHA256()
            else:
                continue
    elif (inputAlgorithm == "AES"):
        print("\nYou have chosen AES!\n");
          print("The Advanced Encryption Standard or AES is a symmetric key algorithm. AES is
the successor algorithm to DES. DES is also like AES but it was easier to crack.")
        print("It is used to encrypt data.")
        response = input("Do you want to learn more about how AES works? Respond yes or no: ")
        if response[0] == "y" or response[0] == "Y":
                print("AES is a symmetric algorithm which means that it uses the same key to
encrypt and decrypt the text.")
                print("AES creates blocks of data from the user inputted text. It then performs
numerous transformations to this blocks such as flipping them and shifting the rules. This is
based from the keys. The algorithm then will note the transformations that it did it the
blocks and then it will create the ciphertext.")
            response3 = input("Do you want to try out AES? Respond yes or no: ")
            if response3[0] == "y" or response[0] == "Y":
                print("Great, let's try out AES!")
                returnS = AES()
                while returnS == 1:
                    returnS = AES()
            else:
                continue
        else:
            response2 = input("Do you want to try out AES? Respond yes or no: ")
            if response2[0] == "y" or response[0] == "Y":
                print("Great, let's try out AES!")
                returnS = AES()
                while returnS == 1:
                    returnS = AES()
            else:
                continue
    elif (inputAlgorithm == "EC"):
        print("\nYou have chosen Elliptic Curve!\n");
         print("Elliptic curve cryptography is one of the procedures used in generating public
and private keys to sign data. It uses elliptic curves to generate keys")
          response = input("Do you want to learn more about how Elliptic Curve works? Respond
yes or no: ")
        if response[0] == "y" or response[0] == "Y":
                print("Elliptic curve cryptography is slowly becoming the next big thing in
cryptography. It is the next wave of asymmetric algorithms that sign messages. But unlike
other poplar algorithms, Elliptic curve cryptography works with less bits, meaning that it is
more efficent")
              print("It uses the math behind Elliptic curves to generate a public and private
key. Each intersection of 3 points in an elliptic curve signifies a number that could be used
as a key for either the public or private key.")
                  print("A great analogy that the Universal Security study suggests is the
competition to Elliptic curve cryptography (RSA), to break a 228 bit key would take less
energy to boil a teaspoon of water. Elliptic curve cryptography, to break a 228 bit key, would
take more energy to boil all the water on Earth.")
            print("This shows how powerful Elliptic curve cryptography is.")
            response3 = input("Do you want to try out Elliptic Curve? Respond yes or no: ")
            if response3[0] == "y" or response[0] == "Y":
                print("Great, let's try out Elliptic Curve!")
                returnS = EC()
                while returnS == 1:
                    returnS = EC()
            else:
                continue
        else:
            response2 = input("Do you want to try out Elliptic Curve? Respond yes or no: ")
            if response2[0] == "y" or response[0] == "Y":
                print("Great, let's try out Elliptic Curve!")
                returnS = EC()
                while returnS == 1:
                    returnS = EC()
            else:
                continue
    else:
        print("You have inputted an unvalid input, please try again")
```